Amendments to the Specification:

Please add the following new paragraph before the section marked "BACKGROUND OF THE INVENTION," on line 5 of page 1:

--CROSS REFERENCES TO RELATED APPLICATIONS

This application is a continuation of U.S. Patent Application No. 10/076,623, filed February 14, 2002, which is a continuation of U.S. Patent Application No. 09/757,078, filed January 8, 2001, which is a continuation of U.S. Patent Application No. 09/246,015, filed February 5, 1999, issued as U.S. Patent No. 6,307,487, which claims priority to U.S. Provisional Patent Application No. 60/101,473, filed September 23, 1998, now abandoned, the disclosures of which are incorporated herein by reference for all purposes.--

Please replace the paragraph beginning at page 21, line 12, with the following rewritten paragraph:

--In operation, for each received output symbol with key I and value B(I), decoder 155 does the following. Key I is provided to value function selector 210, weight selector 215 and associator 220. Using K and key I, weight selector 215 determines weight W(I). Using K, key I and W(I), associator 220 produces the list AL(I) of W(I) positions of input symbols associated with the output symbol. Optionally, using K and I, value function selector 210 selects value function F(I). Then, I, B(I), W(I) and AL(I), and optionally F(I), are stored in a row of output symbol buffer 405. Value function selector 210, weight selector 215 and associator 220 perform the same operation for decoder 155 as described for encoder 115. In particular, the value function F(I), the weight W(I) and the list AL(I) produced by value function selector 210, by weight selector 215 and by associator 220 in Fig. 5 are the same for the same key I as for the corresponding parts shown in Fig. 4. If K varies from input file to input file, it can be communicated from the encoder to the decoder in any conventional manner, such as including it in a message header.--

Please replace the paragraph beginning at page 22, line 24, with the following rewritten paragraph:

--Reducer 415 scans output symbol buffer 405 and reconstruction buffer 425 to find output symbols that have lists AL(I) that list positions of input symbols that have been recovered. When reducer 415 finds such a "reducible" output symbol with key I, reducer 415 obtains the value **[IS_R]** IS(R) of a recovered input symbol at position R and modifies B(I), W(I) and AL(I) as follows:

B(I) is reset to B(I) XOR IS(R)

W(I) is reset to W(I) - 1

AL(I) is reset to AL(I) excluding R

In the equations above, it is assumed that the value function was an XOR of all the associates' values. Note that XOR is its own inverse -- if that were not the case and another value function was used originally to compute the output symbol, then the inverse of that value function would be used here by reducer 415. As should be apparent, if the values for more than one associate are known, the equivalent of the above equations can be calculated to make B(I) dependent only on any unknown associate values (and adjust W(I) and L(I) accordingly).--

Please replace the paragraph beginning at page 26, line 12, with the following rewritten paragraph:

--A variation of the process shown in Fig. 8(a) is shown in Fig. 8(b). There, instead of performing steps 830, 835, 840, 845 and 850 for each output symbol as it is processed, the values of R' and P can be stored in an execution schedule for later processing. An example of deferred execution processing (865) is shown in Fig. 8(c) including steps referenced as 870, 875, 880 and 885. In this variation, the flowchart shown in Fig. 6 is modified by initializing E to zero in step 605. The deferred processing of the execution schedule can occur after the decoder determines that the received symbols are enough to decode the entire file, **[e.g.]** e.g., at step 640 after it is known that all input symbols are recoverable. In some cases, especially where the input symbols are large, the execution of the schedule could be deferred until the input file, or portions thereof, are needed at the receiver.--

Please replace the paragraph beginning at page 26, line 22, with the following rewritten paragraph:

--In either variation, **[i.e.]** i.e., in either Fig. 8(a) or Fig. 8(b), at step 855 the output symbols that still have input symbol P as an associate are modified to reflect that the input symbol has been recovered. The row numbers of these output symbols in OSDS 505 are stored in RL(P). For each row number R" in RL(P), WEIGHT(R") is decremented by one and P is XORed into XOR_POS(R") to reflect the removal of the input symbol in position P as an associate of the output symbol in row R" of OSDS 505. If this modification causes the output symbol in row R" of OSDS 505 to become weight one, i.e., WEIGHT(R") = 1, then this output symbol is added to the decodable set by setting OUT_ROW(S) to R", IN_POS(S) to XOR_POS(R"), and incrementing S by one. Finally, the space used for storing row numbers of output symbols on list RL(P) is returned to free space (860), and processing continues at step 805.--

Please replace the paragraph beginning at page 32, line 1, with the following rewritten paragraph:

--Figs. 16-19 shows a snapshot of an example run of the process described in Fig. 15. In this example, six output symbols (16030, 16035, 16040, 16045, 16050, 16055) have been received with associates (16000, 16005, 16010, 16015, 16020, 16025) indicated as shown by the arrowed lines in Fig. 16. Initially, output symbols with values A, D, A⊕B⊕D⊕F, C, E⊕F and A⊕B (the "⊕" operation being an XOR operation) are received and stored in OSDS 505 as shown in Fig. 17. The row number in OSDS 505 is stored in ROW_LIST in the row corresponding to the weight of the output symbol, as shown in Fig. 18. The output symbols of weight one are in row 0, in row 1, and in row 3 of OSDS 505. Thus, ROW_LIST(0), which corresponds to output symbols of weight WT_VAL(0) = 1, contains row numbers 0, 1 and 3, as shown in Fig. 18. Similarly, ROW_LIST(1) contains 4 and 5 and ROW_LIST(3) contains 2.--

Please replace the paragraph beginning at page 32, line 11, with the following rewritten paragraph:

--At this point in the process, the first five output symbols in order of increasing weight have been processed, the sixth output symbol in row 2 of OSDS 505 has been processed by receive organizer 520 and this sixth output symbol is just about to be processed by recovery processor 525. Output symbols in rows 0, 1, 3 and 5 have already been added to the schedule to eventually recover input symbols in positions 0, 3, 2 and 1, respectively. The output symbol in row 4 of OSDS 505 has two associates at positions 4 and 5 that have not as yet been recovered, and thus there are links from positions 4 and 5 in ISDS 510 back to row 4 in OSDS 505. The output symbol in row 2 of OSDS 505 has four associates in positions 0, 1, 3, and 5. The three associates in positions 0, 1 and 3 have already been marked as recovered, and thus there is no link from them back to row 2 (they caused the weight of this output symbol to be reduced from 4 to 1, which will trigger the recovery of the remaining input symbols in positions 4 and 5 once recovery processor 525 is executed). The associate in position 5 has not been recovered, and thus the receive organizer 520 added a link from position 5 in ISDS 510 to row 2 in OSDS 505. This is all shown in Fig. 17. Thus, at this point in the process, a total of only three links from input symbols back to output symbols which have them as associates are in use. This compares favorably with the straightforward implementation that uses a link from every input symbol to every output symbol having it as an associate. In this example, there [is] <u>are</u> eleven possible such links.--

Please replace the paragraph beginning at page 32, line 30, with the following rewritten paragraph:

--In general, the savings in storage space for links is dramatically reduced when using the process described in Figs. 14 and 15 over the process described in Fig. 13, e.g., the savings in space is typically a factor of 10 to 15 in link space when the number of input symbols is 50,000. The reason for this reduction is that smaller weight output symbols are more likely to recover input symbols at the beginning of the process then at the end, and heavier weight output symbols are much more likely to recover output symbols at the end of the process then at the beginning. Thus, it makes sense to process the output symbols in order of increasing weight. A further advantage of the process described in Figs. 14 and 15 over Fig. 13 is that the decoding is typically 30% to 40% faster. This is because the smaller weight output symbols are more likely to be used to recover input symbols than the heavier weight output symbols (since the smaller weight output symbols are considered first), and the cost of recovering a particular input symbol directly depends on the weight of the output symbol used to recover it.--

Please replace the paragraph beginning at page 35, line 7, with the following rewritten paragraph:

--The methodology for determining the distributions in one preferred embodiment will now be described. The actual weight distributions used are based on an ideal mathematical distribution. In the ideal weight distribution, the weights W(I) are chosen according to an "ideal" probability distribution. The smallest weight is one and the largest weight is K, where K is the number of input symbols. In the ideal distribution, a weight equal to a value of *i* is chosen with the following **[probability:]** probability p:

for i=1:	p=1/K; and
for i=2,...,K:	p=1/(i(i-1)).

Once a weight W(I) chosen, a list AL(I) of W(I) associated input symbols are chosen independently and uniformly at random (or pseudorandomly, if needed), making sure that all chosen associates are distinct. Thus, the first associate is randomly selected among the K input symbols, each having a probability of 1/K of being selected. The second associate (if W>1) is then randomly selected among the remaining K-1 symbols. The weight probability distribution shown above has the property that if the system behaved exactly as expected, exactly K output symbols would be sufficient to decode and recover all input symbols. This expected behavior for the ideal distribution is shown in Fig. 20 by the solid line. However, because of the random nature of selection of the weights and the associates, and because an arbitrary set of output symbols are used in the decoding process, the process will not always behave that way. An example of an actual behavior for the ideal distribution is shown in Fig. 20 by the dotted line. Hence, the ideal weight distribution must be modified somewhat in practice.--

Please replace the paragraph beginning at page 36, line 7, with the following rewritten paragraph:

--More specifically, the modified weight distribution is as follows:

for *i*=1:	p=n·R1/K;
for *i*=2,…, (K/R2 - 1):	**[p=n·]**p=n/(*i*(*i*-1)(1-*i*R2/K));
	and
for *i*=K/R2,…,K:	p=n·HW(*i*)

where K is the number of input symbols, R1 and R2 are tunable parameters and n is a normalization factor used so that the p values all sum to one.--

Please replace the paragraph beginning at page 37, line 6, with the following rewritten paragraph:

--The number of output symbols generated and sent through the channel is not limited with chain reaction coding as with other coding schemes, since keys need not have a one-to-one correspondence with input symbols and the number of different values of I is not limited to some ratio of input symbols. Therefore, it is likely that even if the decodable set goes empty before the input file is reconstructed, the decoding process will not fail, since the decoder can gather as many more output symbols as needed **[go]** to get at least one more output symbol of weight one. When that output symbol of weight one is received, it populates the decodable set and, by the chain reaction effect, might cause reduction of previously received output symbols down to weight one so that they can, in turn, be used to reconstruct input symbols.--

Please replace the paragraph beginning at page 38, line 18, with the following rewritten paragraph:

--Chain reaction decoding has a property that if the original file can be split into K equal-sized input symbols and each output symbol value is the same length as an input symbol value, then the file can be recovered from K + A output symbols on average, where A is small compared to K. For example, A might be 500 for K=10,000. Since the particular output symbols are generated in a random or pseudorandom order, and the loss of particular output symbols in transit is **[assumed random]** arbitrary, some small variance exists in the actual number of output symbols needed to recover the input file. In some cases, where a particular collection of K + A packets are not enough to decode the entire input file, the input file is still recoverable if the receiver can gather more packets from one or more sources of output packets.--

Please replace the paragraph beginning at page 41, line 28, with the following rewritten paragraph:

--In one aspect, the chain reaction coding process described above performs the digital equivalent of a holographic image, where each part of a transmission contains an image of the transmitted file. If the file is a megabyte file, a user can just tap into a transmitted stream to obtain any **[random]** arbitrary megabyte's worth of data (plus some extra overhead) and decode the original megabyte file from that **[random]** megabyte.--